Models of Agent Interaction based on Modal Logics

Matteo Baldoni *, Cristina Baroglio, Elisa Marengo, Viviana Patti and Claudio Schifanella

Dipartimento di Informatica, Università degli Studi di Torino, c.so Svizzera 185, I-10149 Torino, Italy
E-mail: {baldoni,baroglio,emarengo,patti,schi}@di.unito.it

Abstract. The specification of interaction and of the related forms of reasoning is crucial in the research area of multi-agent systems and in many application areas. This article summarizes the activities and the achievements obtained by the authors in the last years. Two are the main research lines, respectively leading to the proposal of a mentalistic approach to the specification of agent interaction policies, based on the class of grammar logics, and the proposal of a constraint-based representation of regulative and declarative patterns of interaction inside commitment-based protocols.

Keywords: Interaction protocols, multimodal logics, agents, web services, semantic web, e-learning, commitments.

1. Introduction

Modal logics are widely used in Artificial Intelligence for representing knowledge and beliefs together with other attitudes like, for instance, goals, intentions and obligations. Moreover, modal logics are well suited for representing dynamic aspects in agent systems and, in particular, for formalizing reasoning about actions and time [22,26,41]. In this context, one of the main recent research lines concerns the specification of interaction and the forms of reasoning that can be applied to it, with a particular attention to the verification of properties of the interaction itself and of the interacting agents [41]. Models of interaction have, in fact, as diverse application domains as e-learning, web service selection and composition, coordination based on choreographies, financial protocols.

In the last years we worked both on a mentalistic approach to agent interaction and on a commitment-based approach. For what concerns the former, we studied a logical and computational framework for representing agent interaction policies in a declarative way [2,14,31], that is based on a class of normal multimodal logic called grammar logics. The framework allows for reasoning on the effects of adopting a protocol role from the subjective perspective of the individual agent, and it was successfully applied in the context of e-learning and of web services, in various national and international research projects, like MASIVE, SVP, and REWERSE. The added value in both applications is that this reasoning approach allows to supply personalized services to the users and support them both in the construction of ad hoc curricula and in obtaining a flexible and customized composition of services [4,10].

More recently, we studied a social approach to the representation of interaction protocols, with a particular focus on the representation of declarative patterns of interaction, having a regulative nature [5]. Social approaches allow reasoning on the expectations of the participants to the interaction from a global perspective, without the need of performing any introspection on the agents’ mental states but only exploiting the public commitments and, therefore, only the observational properties expressed by the protocol. Patterns of interaction are specified by means of constraints among commitments by using a modal language based on linear time temporal logic (LTL) [22].

*Corresponding author. E-mail: baldoni@di.unito.it.
2. Grammar Multimodal Logics

[13,2] present a class of normal multimodal logics, called grammar logics. The class is characterized by a set of inclusion axioms of the form:

\[ [p_1] \ldots [p_n] \varphi \supset [s_1] \ldots [s_m] \varphi \ (n > 0; m \geq 0) \]  

(1)

where the \( p_i \)'s and \( s_j \)'s are modalities. This class includes some well-known modal systems such as K, K4, S4, and their multimodal versions. Differently from other logics, such as those in [26], these systems can be non-homogeneous, that is, the modal operators are not restricted to belong to the same system. They can also contain some interaction axioms, which means that the modal operators are not restricted to be mutually independent.

This class of logics has been introduced by Fariñas del Cerro and Penttonen in [23] to simulate the behavior of formal grammars. Given a formal grammar, each terminal and non-terminal symbol is associated to a different modality. Furthermore, an inclusion axiom \([p_1] \ldots [p_n] \varphi \supset [s_1] \ldots [s_m] \varphi\) is defined for each production rule \( p_1 \cdots p_n \rightarrow s_1 \cdots s_m\). Grammar logics have many applications; for instance, they have been used in the study of description logics [35,27].

Grammar logics have interesting computational properties, that led to the development of an agent programming language, introduced in the next section. The foundation is the analytic tableau calculus, that is presented in [13,2]. The calculus is parametric w.r.t. the logics of this class; in particular, it can deal with non-homogeneous multimodal systems with arbitrary interaction axioms of form (1). The calculus is a prefixed tableaux extension of those in [29,25]. Prefixes are given atomic names and their accessibility relations are explicitly represented by means of a graph. The key idea is using the characterizing axioms of the logic as "rewrite rules", which create new paths among worlds in the counter-model construction. [23] proves the undecidability of the whole class of grammar logics by showing the equivalence of testing whether a word is generated by the formal grammar, and proving a theorem in the logic. The work in [13,2] refines this result: the authors prove the undecidability of modal systems, that are based on context-sensitive and context-free grammars, while right regular grammars are proved to be decidable. This is done by using an extension of the filtration methods by the Fischer-Ladner closure for modal logics. Further refinements can be found in [21,3].

3. Dynamics in LOGic

A notable subclass of grammar logics is the one that includes axiom schemas of the form

\[ \langle s_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle \varphi \]  

(2)

In this case, the rewriting rules for describing accessibility relations become similar to a Prolog goal-directed proof procedure. This characteristic brought, as a natural consequence, the definition of a computational counterpart which developed into the programming language Dynamics in LOGic [14,28].

Dynamics in LOGic is an agent programming language, which is based on a logical theory for reasoning about actions and change in a modal logic programming setting. An agent’s behavior is described in a non-deterministic way by giving the set of actions that it can perform. Specifically, it is given by a domain description, which includes: a) action and precondition laws, describing the atomic world actions, that the agent can perform; b) a set of sensing axioms, describing the agent’s atomic sensing actions; c) a set of procedure axioms, describing the agent’s complex behavior. Typically, each atomic action has preconditions to its application (that decide if the action is executable) and effects due to its application. Effects can also be subject to additional conditions. For instance, the precondition to the executability of the action “pay by credit card” is that the payer owns a valid credit card. A conditional effect of this action may be "to be notified by SMS about the payment". This effect will become true only if the owner previously subscribed the proper service.

Given this view of actions, the problem of reasoning can be interpreted as the act of building or traversing a sequence of state transitions. Technically speaking, a state is a set of fluents, i.e., properties whose truth value can change over time. Such properties encode the information that flows during the execution of the agent’s behavior: for instance, if a buyer communicates to pay by credit card, this information will be included in the state of the merchant as a fluent. In general, we cannot assume that the value of all fluents in a state is known. Dynamics in LOGic offers the means for representing that some fluents are unknown, as well as the tools for reasoning about the execution of actions on incomplete states. In fact, it includes an epistemic operator \( B \) to represent the beliefs an agent has about the world: \( B f \) means that the fluent \( f \) is known to be true, \( B \neg f \) means that the fluent \( f \) is known to be false. A fluent \( f \) is undefined when both \( \neg B f \) and
An agent’s behavior can be specified by means of procedures, i.e., Prolog-like clauses built upon other actions. Formally, a complex action is a collection of inclusion axiom schemas of form (2), where \( s_q \) is a procedure name and the \( p_i \)’s are procedure names, atomic actions, or test actions (? \( f \)). Procedure definitions may be recursive and procedure clauses can be executed in a goal-directed way, similarly to standard logic programs. The language includes a communication kit [31], which allows the specification of communicative behaviors [9]. This consists of a predefined set of speech acts, that are modeled in terms of action and procedures which specify the behavior of agents that are involved in a communication. A communicative action modifies not only the beliefs of the executor about the world but also its beliefs about the interlocutor’s mental state.

It is possible to reason about domain descriptions and formalize the temporal projection and the planning problems by means of existential queries of form:

\[
\langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_m \rangle Fs
\]  
(3)

where each \( p_k \), \( k = 1, \ldots, m \) may be an (atomic or complex) action executed by the agent and \( Fs \) is a conjunction of fluents. Checking if a query of form (3) succeeds, corresponds to answering the question “Is there an execution trace of \( p_1, \ldots, p_m \) which leads to a mental state where \( Fs \) holds?”. In case all the \( p_k \)’s are atomic actions, it amounts to predict if the condition of interest will be true after their execution (temporal projection). In case complex actions are involved, the execution trace that is returned in the end is a plan to bring about \( Fs \). The procedure definition constrains the search space. The plan can be linear or conditional because whenever a sensing action is involved, during the reasoning, all of their possible outcomes must be considered.

An interpreter for the language was implemented in Sicstus Prolog [28]: it uses a goal-directed proof procedure, based on negation as failure, to prove queries of form (3). This implementation also allows the language to be used as an ordinary programming language for executing procedures which specify the behavior of an agent.

### 4. Reasoning about Web Services

Distributed applications over the World-Wide Web have obtained wide popularity. Uniform mechanisms have been developed for handling computing problems, which involve a large number of heterogeneous components, that are physically distributed and that interoperate. These developments coalesced around the web service paradigm. A web service can be seen as a component that is available over the web [1]. Each service has an interface that is accessible through standard protocols and that describes the interaction capabilities of the service. Compositions of services are often expressed, in the literature, as patterns of interaction, represented by means of a choreography language, like WS-CDL [42]. The description is done from a global point of view, encompassing the expected behavior of all the participants.

One of the key ideas behind web services is that services should be amenable to automatic retrieval, thus facilitating their re-use. To this aim, there are proposals, such as e.g. OWL-S [30] and WSMO [24], to enrich the service descriptions with a semantic layer. Typically, the semantic annotation concerns the inputs, outputs, preconditions and effects of the service. Inputs and outputs are usually expressed by ontological terms, while preconditions and effects are often expressed by means of logic representations. Services can be given an action-based semantic representation by using the Dynamics in LOGic language [9,10]: each service operation can, in fact, be described in terms of its preconditions and effects.

Semantic annotations alone, however, are not sufficient to accomplish the selection and composition tasks. In this case, it becomes useful to introduce a notion of goal, which opens the way to the introduction of another abstraction, that of agent [33,40,10]. Agents show not only the ability of performing goal-driven forms of reasoning, but they also show autonomy and proactivity, which are helpful characteristics when dealing with open environments, allowing for instance a greater fault tolerance [15,16,34,18]. Moreover, the matchmaking techniques that are currently used to perform the service selection work on a single operation at a time. However interaction specifications like choreographies provide a precise context for the selection of services, that should be taken into account during the (semantic) matching process. The agent paradigm offers proper abstractions to naturally articulate a model of the interaction among services by distinguishing two different levels: the choreography
level, where shared patterns of interaction among a set of services are described from a global and public perspective, and the interaction policy level, where the interactive behavior of the single service is represented from a local perspective. The Dynamics in LOGic language has been used to represent both service interaction policies as well as choreography roles. By exploiting the reasoning mechanisms, it is possible to overcome the limits of the classical semantic matchmaking techniques [11].

A first improvement is presented in [9], where web service selection and composition is done on the base of the goals expressed by a user and a semantic description of the service interaction policies.

When choreographies are also represented [10, 11], it becomes possible to verify if the adoption of a certain role allows the achievement of a goal. More specifically, given a logic-based representation of the choreography role, interpreted as an agent’s behavior, and a representation of the agent’s goals, the selection and composition problems amount to decide if the role fits the goals. The reasoning task becomes “Is it possible to play this role in such a way that the goals are achieved?”, which is exactly the kind of problem. Dynamics in LOGic was developed to tackle. In this process, the match compares the abstract requirements specified in the role with the actions (capabilities) that the agent can actually execute when it will enact the role. The matching process guarantees that the goals that can be achieved before the substitution will be achievable also after.

The work in [11] generalizes and extends the previous results by tackling the problem of flexible matchmaking [39], that amounts to select services whose preconditions and effects do not match with the requirement exactly, in the context of choreographies. Given a role in a shared interaction protocol, a service can compute a set of execution traces that allow the achievement of its goal, and it can identify the conservative substitutions [11] for all the involved operations. Intuitively, the match is conservative if it preserves the goals even though the selected service operations are not identical to the specified requirements. The work identifies a set of conservative re-use ensuring matches from those defined in the literature [39].

On the other hand, services are often used not in an individual way but jointly, for executing tasks that none of them alone can accomplish. Each service, that is involved in a choreography, has its own goals, that it tries to pursue. The research question is whether it is possible for this team of autonomous services to reach an agreement about their possible executions, and find a joint working plan, so that in the team all of the services will achieve their personal goals. In [8], the services that will play the roles of the choreography can restrict the set of execution traces by keeping only those traces which allow for the achievement of their goals. Once this set is known by all the involved services, whatever the initiator of the interaction will be, each partner knows how to behave so to allow the mutual achievement of their personal goals.

5. Curricula planning and validation

Dynamics in LOGic has been used to implement an adaptive tutoring system [12] with a multi-agent architecture, that can produce personalized study plans and that can validate study plans built by a user. A key feature that allows the tutoring system agents to adapt to users is their ability to tackle mental attitudes, such as beliefs and intentions. The agent can adopt the user’s learning goal and find a way for achieving it, which fits the specific student’s interests and takes into account his/her current knowledge.

A natural evolution of this work opened the way to the activity carried on in the REWERSE NoE [4], where a layered architecture is proposed that combines a Semantic Web approach to resource annotation with a declarative representation of constraints in LTL [22] for representing learning resources, the domain model, the learner, and pedagogical constraints. The representation allows the construction of personalized curricula by means of planning techniques as well as different kinds of interconceptual, post-construction verifications, all of which can be realized by means of model checking techniques. In particular, the work proposes a graphical language called DCML, grounded on LTL, for the definition of the constraints that a given curriculum of study must respect. By means of this language it is possible to express temporal constraints on the order in which competences are to be acquired, requirements on the learner’s initial knowledge as well as to express which competences are mandatory. A prototype implementation based on the Personal Reader framework was developed, where the above techniques were used for suppling to the users personalization functionalities, implemented in a service-oriented fashion.

Commitment-based protocols [19,37,43] are one of the most successful approaches to the specification of interaction protocols. The greatest advantages of using them instead of other approaches to interaction are that they do not over-constrain the agents’ behavior, by imposing unnecessary orderings on the execution of the shared actions, and that, by giving a shared meaning to the social actions, they allow working on actual knowledge rather than on beliefs about each others’ mental state. Commitments have a “regulative” nature in the sense that agents are bound to make the condition in their commitments true. Nonetheless, commitment protocols do not suit well those situations where the evolution of the social state is constrained by laws, preferences or habits, because they do not allow the specification of legal patterns of execution, although this kind of constraints makes sense in many practical situations, as noticed also in [38]. Contrarily to constitutive rules, which define new forms of behavior, these patterns regulate antecedently existing forms of behavior [36]. Therefore, they regulate the social reality, defined by the constitutive rules.

The work in [5,6] proposes a model for commitment interaction protocols that separates the constitutive and the regulative parts and supplies first-class languages for representing both in a flexible way. In particular, for the constitutive specification it adopts [20,17], while for the regulative specification it proposes the use of constraints among commitments and a language, 2CL (standing for “Constraints among Commitments Language”), that allows the specification of different kinds of such constraints. 2CL is grounded on the LTL modal logic and consists of seven kinds of constraints. The names of its operators and their graphical format are inspired by ConDec [32] and by DCML [4].

For instance:

\[ C(i, p, \text{assigned\_task}(i, p)) \rightarrow (\neg \text{refused\_task}(p, i) \lor C(p, i, \text{solved\_task}(p, i))) \]

expresses the fact that \( p \) is not allowed to refuse a task \( i \) nor it is allowed to commit to solve it before \( i \) has taken a commitment, stating its intention to assign the task to \( p \). In LTL this constraint is expressed by:

\[ \neg(-\text{refused\_task}(p, i) \lor C(p, i, \text{solved\_task}(p, i))) \lor C(i, p, \text{assigned\_task}(i, p)). \]

A clear separation of the constitutive from the regulative specification brings many advantages, mostly as direct effects of the obtained modularity: easier re-use of actions in different contexts, easier customization on the protocol, easier composition of protocols [7]. The declarative nature of the language preserves the flexibility that is typical of commitment-based protocols and does not compromise the autonomy of agents, which would be free to decide how to act and to take advantage of opportunities, that arise along the interaction.

7. Acknowledgements

The authors would like to thank all people who contributed to this research over the last years, in particular Laura Giordano, Alessandro Chiarotto and Laura Torasso. Special thanks to Alberto Martelli, who worked with us on many of the presented topics and who has always been an inspiring guide.

References


